CompSci Journal
Double Blind Peer Reviewed, Open Access International Research Journal.
ISSN: 3067-3089 | *Website:* https://csjour.com | *Volume-1, Issue-3 | June - 2025*

Original Article

Tensors, Data Structures, and Tensor Algebra in Programming: A Comprehensive Study

M. A. Jadhav[1], S. P. Thorat[2], P. N. Deorukhakar[3]
[1]Department of Computer Studies, Vivekanand College, Kolhapur, (An Empowered Autonomous Institute) Maharashtra, India
[2]Department of Mathematics, Vivekanand College, Kolhapur, (An Empowered Autonomous Institute) Maharashtra, India
[3]Department of Computer Science, Shivaji University, Kolhapur, Maharashtra, India

Manuscript ID:
CSJ-2025-010301

Volume 1

Issue 3

Pp. *1-6*

June 2025

ISSN: 3067-3089

Submitted: 15 May. 2025

Revised: 25 May. 2025

Accepted: 15 June. 2025

Published: 30 June. 2025

Correspondence Address:
M. A. Jadhav, Department of Computer Studies, Vivekanand College, Kolhapur, (An Empowered Autonomous Institute) Maharashtra, India
Email: ma.ja1984@gmail.com

Quick Response Code:

Web. https://csjour.com/

DOI : 10.5281/zenodo.17710420

DOI Link:
https://doi.org/10.5281/zenodo.17710420

Abstract

Tensors, as multidimensional extensions of scalars, vectors, and matrices, are foundational to numerous scientific and computational domains, including physics, quantum mechanics, deep learning, and signal processing. This paper provides a comprehensive exploration of tensor theory, algebraic operations, tensor calculus, and their practical implementation across a diverse range of programming languages, including C, C++, C#, Java, Python3, MATLAB, R, Julia, Rust, Go, and Scala. We investigate the properties of covariant, contravariant, and mixed tensors, including their transformation behaviors and symmetry features. Key tensor operations, including as outer and inner products, tensor contraction, Kronecker products, and the t-product, are thoroughly investigated with mathematical formulations and related code examples. Tensor calculus concepts, including the metric tensor, Christoffel symbols, and covariant derivatives, are introduced, with a focus on their applications in physics and engineering. The study also delves into advanced tensor decompositions—CP, Tucker, and Tensor-Train—demonstrating their utility in machine learning, biomedical engineering, and quantum computing. By bridging theoretical tensor mathematics with practical programming, this work serves as a vital resource for researchers and practitioners, offering insights into tensor learning, structured algebra, and hardware-accelerated computations. Future directions include standardizing tensor libraries and advancing tensor calculus in emerging fields like quantum machine learning.

Keywords*: Tensors, Tensor Algebra, Tensor Calculus, Covariant, Contravariant, Outer Product, Inner Product, Kronecker Product, t-Product, CP Decomposition, Tucker Decomposition, Tensor-Train, Programming Languages, Tensor Learning*

Introduction

In domains such as physics, computer vision, quantum computing, and machine learning, tensors allow the representation of complex, structured data by generalizing scalars (0D), vectors (1D), and matrices (2D) to higher-dimensional arrays [1]. Tensors in deep learning stand in for inputs, weights, and neural network activations. They replicate the strain, tension, and curvature of space time in general relativity. Despite their importance, tensors are often underutilized in general-purpose programming due to their abstract mathematical nature and inconsistent language support. This paper provides a comprehensive exploration of tensor theory, algebra, calculus, and their practical implementation. **Our objectives are:**

To clarify the transformation properties of the three types of tensors (covariant, contravariant, and mixed).
To express core tensor operations and calculus with mathematical formulations and code examples.
To compare tensor support across a broad range of programming languages, including emerging ones like Rust, Go, and Scala.
To explore advanced tensor decompositions and their applications in real-world domains.
To survey recent advances in tensor learning, structured algebra, and hardware acceleration.

Methodology

This study employs a multidisciplinary approach to investigate tensors, combining theoretical analysis, mathematical formulations, and practical programming implementations. The methodology is structured as follows:

How to cite this article:
Jadhav, M. A., Thorat, S. P., & Deorukhakar, P. N. (2025). Tensors, Data Structures, and Tensor Algebra in Programming: A Comprehensive Study. CompSci Journal, 1(3), 1–6.
https://doi.org/10.5281/zenodo.17710420

1. Literature Review: An extensive review of existing literature on tensor theory, algebra, and calculus was conducted, drawing from seminal works in mathematical physics, machine learning, and computational science (e.g., Schutz [1], Kolda and Bader [2]). This informed the theoretical framework and identified gaps in tensor applications across programming languages.
2. Tensor Theory and Classification: Tensors were categorized based on transformation properties (covariant, contravariant, mixed) and symmetry (symmetric, antisymmetric). Mathematical formulations for tensor transformations under coordinate changes were derived to ensure clarity and rigor.
3. Tensor Operations Analysis: Core operations (outer/inner products, tensor contraction, Kronecker products, t-product) were formalized mathematically and implemented in multiple programming languages. Libraries such as NumPy, PyTorch, TensorOperations, and Eigen were used to develop code examples, ensuring practical applicability.
4. Tensor Calculus Exploration: Concepts like the metric tensor, Christoffel symbols, and covariant derivatives were studied using symbolic computation tools (e.g., SymPy) and applied to physics and engineering scenarios.
5. Programming Language Comparison: Tensor support was evaluated across 11 languages (C, C++, C#, Java, Python, MATLAB, R, Julia, Rust, Go, Scala) through a comparative analysis of data structures and libraries, focusing on usability and performance.
6. Tensor Decompositions and Applications: Advanced decompositions (CP, Tucker, Tensor-Train) were implemented using tools like TensorLy and TensorToolbox, with applications analyzed in machine learning, signal processing, and quantum computing.
7. Recent Advances and Synthesis: Emerging trends, such as structured tensor algebra, tensor learning, tubal algebra, and hardware acceleration, were explored through recent publications and practical experiments, synthesizing theoretical and computational insights.
This methodology ensures a robust, holistic study, blending rigorous mathematics with practical programming to advance tensor research and application.

Tensor Theory and Types

Tensors are mathematical objects that transform systematically under coordinate changes, making them invariant across different reference frames [1]. They are classified based on their transformation properties:

Covariant tensors (indices down, e.g.,T_{ij}): Transform with the basis, representing quantities like gradients or differential forms. Common in physics for describing fields.

Contravariant tensors (indices up, e.g.,T^{ij}): Transform inversely to the basis, representing quantities like velocities or displacements

Mixed tensors (e.g.,T^{i}_{j}): Combine covariant and contravariant indices, used in transformations like linear maps or stress tensors in mechanics.

A tensor of order N (or rank N) is a multidimensional array with N indices. The transformation under a coordinate change from x to x' for a rank-2 covariant tensor T_{ij}

$$T'_{ij} = \left(\frac{\partial x^k}{\partial x'^i}\right)\left(\frac{\partial x^l}{\partial x'^j}\right)T_{kl}$$

This guarantees that physical rules stay the same across coordinate systems [1]. In programming, tensors are represented as multidimensional arrays, with efficient manipulation relying on specialized libraries.

Tensors are also categorized by their symmetry properties, such as symmetric (e.g.,$T_{ij} = T_{ji}$) or antisymmetric tensors, which are critical in applications like electromagnetism (e.g., the electromagnetic field tensor$F_{\mu\nu}$) [11]. Understanding these properties is essential for implementing tensors in computational frameworks.

Core Tensor Operations

Matrix and vector operations are extended to higher dimensions by tensor operations. Below, we describe key operations with mathematical formulations and code examples across multiple languages.

Outer and Inner Products

The outer product of vectors $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ produces a rank-2 tensor$T \in \mathbb{R}^{m\times n}$:

$$T_{ij} = u_i\, v_j$$

The inner product (dot product) yields a scalar: $\langle u, v\rangle = \Sigma_i\, u_i\, v_i$

Python Example (NumPy):

```
import numpy as np
u = np.array([1, 2, 3])
v = np.array ([4, 5, 6])
Outer = np.outer (u, v) # [[4, 5, 6], [8, 10, 12], [12, 15, 18]]
inner = np.dot(u, v)   # 32
```

Scala Example (Breeze):

```
Import breeze.linalg.DenseVector
val u = DenseVector (9.0, 12.0, 13.0)
val v = DenseVector(-6.0, 23.0, 16.0)
val outer = u * v.t  // Outer product
val inner = u dot v  // 32.0
```

CompSci Journal
Double Blind Peer Reviewed, Open Access International Research Journal.
ISSN: 3067-3089 Website: https://csjour.com | Volume-1, Issue-3 | June - 2025

Tensor Contraction

Tensor contraction lowers the tensor's order by two by adding together shared indices. When indices j and k are contracted for a rank-4 tensorT_{ijkl}, the result is: $S_{il} = \Sigma_{j,k} T_{ijkl}$

Relativity and machine learning both employ this operation to reduce dimensionality [2].

Julia Example:

Using Tensor Operations
T = rand (2, 3, 3, 2) # Rank-4 tensor
@tensor S[i,l] := T[i,j,j,l] # Contract j,k

Rust Example (ndarray):

Use ndarray::{Array4, s};
Let t = Array4 ::< f64>: zeros ((2, 3, 3, and 2));
Let s = t.sum_axis (Axis (1)).sum axis (Axis (1)); // Contract j,k

3 Kroenke Product

The Kronecker product $A \otimes B$ of matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ generates a block matrix of size $mp \times nq$, generally used in quantum computing and signal processing [2].

MATLAB Example:

A = [1, 2; 3, 4];
B = [0, 5; 6, 7];
C = kron(A, B);

Go Example (Gonum):

Package main
Import "gonum.org/v1/gonum/mat"
Func main () {
X := mat.NewDense(2, 2, []float64{8, 2, 6, 4})
Y: = mat.NewDense (2, 2, [] float64{10, 5, 16, 17})
Var z mat. Dense z.Kron(x, y)
}**.4 t-Product**

The t-product, introduced by Kilmer et al. [3], operates on third-order tensors in the Fourier domain, enabling tensor singular value decomposition (SVD) and compression: $C = A * B$

This is particularly useful for video processing and hyperspectral imaging [4].

MATLAB Example (TensorLab):

X = rand(7, 5, 3); Y = rand (6, 8, 3);
Z = tprod(X, Y); % t-product using TensorLab

Tensor Calculus

Tensor calculus extends tensor algebra to differential operations, critical for applications in physics, engineering, and machine learning [11]. The following table summarizes key tensor calculus operations.

Table 1: Tensor Calculus Operations

Operation	Description
Metric Tensor	Defines geometry of space, e.g., $ds^2 = g_{ij}\, dx^i\, dx^j$
Christoffel Symbols	Describe basis vector changes, e.g., Γ_{ij}^k
Covariant Derivative	Generalizes differentiation, e.g., $\nabla_k v^i$
Riemann Curvature	Measures spacetime curvature, e.g., R_{jkl}^i

1 Metric Tensor

The metric tensor g_{ij} defines the geometry of a space, used in general relativity to compute distances:

$$ds^2 = g_{ij}\, dx^i\, dx^j$$

In Euclidean space, $g_{ij} = \delta_{ij}$ (Kronecker delta).

Python Example (SymPy):

from sympy import Matrix, diag
g = diag (1, 1, 1) # Euclidean metric tensor

2 Christoffel Symbols

Christoffel symbols Γ_{ij}^k describe how basis vectors change in curved spaces, used in geodesic equations:

$$\Gamma_{ij}^k = \left(\frac{1}{2}\right) g^{kl} \left(\frac{\partial g_{il}}{\partial x^j} + \frac{\partial g_{jl}}{\partial x^i} - \frac{\partial g_{ij}}{\partial x^l}\right)$$

These are computed symbolically or numerically in libraries like SymPy or Tensor Flow [6].

Python Example (SymPy):

```
From sympy.diffgeom import Metric Tensor, Tensor Product
From sympy import symbols
X, y = symbols ('x y')
g = Matrix([[x, 0], [0, y]])
Metric = Metric Tensor(g, (x, y))
christoffel = metric.connection ()
```

Covariant Derivative

The covariant derivative $\nabla_k \, v^i$ generalizes differentiation to curved spaces, ensuring tensorial transformation:

$$\nabla_k \, v^i = \frac{\partial v^i}{\partial x^k} + \Gamma^i_{kl} \, v^l$$

This is critical for modeling physical systems like fluid dynamics.

Programming Language Support

Efficient tensor manipulation requires robust data structures and libraries. The following table compares tensor support across 11 programming languages, including emerging ones like Rust, Go, and Scala.

Table 2: Tensor Support across Programming Languages

Language	Data Structure	Libraries
C	Static arrays	BLAS, LAPACK
C++	Eigen, xtensor	Eigen, Armadillo
C#	T[,], T[][]	Math.NET Numerics
Java	double[][]	ND4J, DeepLearning4J
Python	ndarray	NumPy, PyTorch, TensorFlow
MATLAB	Native arrays	Built-in, TensorLab
R	array(), tensorA	rTensor, tensor
Julia	Array{T,N}	Tensor Toolbox, Tensor Operations
Rust	ndarray	ndarray, tract
Go	Slice of slices	Gonum, Gota
Scala	DenseMatrix	Breeze, Smile

1. Language-Specific Implementations

Python: NumPy and PyTorch offer intuitive tensor operations, widely used in machine learning:

```
Import torch
T = torch.randn (2, 3, 4) # 3rd-order tensor
T_permuted = T.permute(1, 0, 2)  # Reorder dimensions
```

Julia: Native multidimensional arrays and tensor Operations enable high-performance computing:

```
Using tensor Operations
A = rand(2, 3, 4); B = rand(3, 4, 5)
@tensor C[i,j,k] := A[i,m,n] * B[m,n,k]
```

Rust: The ndarray crate supports tensor operations, with tract for machine learning:

```
Use ndarray::Array3;
Let a = Array3 ::< f64>: zeros ((2, 3, and 4));
Let b = Array3 ::< f64>: ones ((3, 4, and 5));
```

Go: Gonum provides matrix and limited tensor operations:

```
Package main
Import "gonum.org/v1/gonum/mat"
func main() { A: = mat.NewDense (2, 2, [] float64 {1, 2, 3, 4})}
```

Scala: Breeze supports tensor-like operations via DenseMatrix:

```
Import breeze.linalg.DenseMatrix
val A = DenseMatrix((1.0, 2.0), (3.0, 4.0))
```

C++: Eigen and xtensor provide flexible tensor operations, though they require more boilerplate:

```
#include <xtensor/xtensor.hpp>
xt::xtensor<double, 3> T = xt::zeros<double>({2, 3, 4});
```

Tensor Decompositions

Tensor decompositions reduce high-dimensional data into interpretable components, critical for data compression and feature extraction [2].

1 CP Decomposition

The CANDECOMP/PARAFAC (CP) decomposition expresses a tensor as a sum of rank-1 components. For a third-order tensor$T \in \mathbb{R}^{I \times J \times K}$:

$$T \approx \Sigma_r = 1^R a_r \circ b_r \circ c_r$$

Where ∘ denotes the outer product. Implemented in TensorLy (Python):

```
From tensorly.decomposition import parafac
Import numpy as np
T = np.random.rand (10, 10, 10)
Factors, _ = parafac(T, rank=5)
```

2 Tucker Decomposition

Tucker decomposition factorizes a tensor into a core tensor and orthogonal factor matrices:

$$T \approx G \times_1 A \times_2 B \times_3 C$$

This is more flexible than CP but computationally intensive [7].

3 Tensor-Train Decomposition

Tensor-Train (TT) decomposition represents a tensor as a sequence of low-rank matrices, ideal for high-dimensional data [12]:

$$T(i_1, \ldots, i_d) \approx G_1(i_1) G_2(i_2) \ldots G_d(i_d)$$

Implemented in Julia's Tensor Toolbox:

```
Using Tensor Toolbox
T = rand(10, 10, 10)
Tt = ttsvd(T, (2, 2, 2))  # Tensor-Train decomposition
```

Applications

Tensor techniques are applied across diverse domains:

Machine Learning: Tensor decompositions for recommendation systems and neural network compression [9].

Signal Processing: EEG/fMRI analysis using CP decomposition for brain activity modeling [8].

Physics: Tensor calculus for general relativity, modeling spacetime curvature [11].

Computer Vision: Video compression via tensor SVD [3].

Quantum Computing: Tensor networks for simulating quantum states [10].

Biomedical Engineering: Tensor-based analysis of MRI data for disease diagnosis [8].

Recent Advances

Recent developments enhance tensor computation efficiency and applicability.

1 Structured Tensor Algebra

The Tensor Algebra Compiler (TACO) optimizes sparse and dense tensor operations, reducing memory and computation costs for large-scale applications [5].

2 Tensor Learning

Tensor-based neural networks leverage low-rank approximations to improve scalability in deep learning, particularly for high-dimensional datasets [6].

3 Tubal Algebra

Tubal algebra, based on the t-product, supports robust compression for hyperspectral imaging and video analysis [4].

4 Hardware Acceleration

GPU and TPU acceleration in frameworks like PyTorch and Tensor Flow significantly speeds up tensor decompositions and neural network training [6].

5 Tensor Calculus in Machine Learning

Recent work integrates tensor calculus with machine learning, using covariant derivatives to optimize neural network training on manifolds [13].

Conclusion

Tensors are indispensable in computational science, bridging theoretical mathematics with practical applications. This paper has provided a comprehensive study of tensor theory, algebra, calculus, programming language support, decompositions, and recent advances. By combining mathematical rigor, code examples across diverse languages, offers a valuable resource for researchers and practitioners. Future work should focus on standardizing tensor libraries, optimizing sparse tensor operations, and exploring tensor calculus in emerging fields like quantum machine learning and geometric deep learning.

Acknowledgment

The authors would like to express their sincere gratitude to Vivekanand College, Kolhapur,
(An Empowered Autonomous Institute) Maharashtra, India for providing the necessary facilities and support to carry out this research. We also Dr. R. Y. Patil, Asst. Prof. Dept. Of Computer Science, Vivekanand College, Kolhapur for their valuable guidance and insightful suggestions during the course of this study.

Financial Support and Sponsorship

The authors did not receive any financial support or sponsorship for the research, authorship, or publication of this article.

Conflicts of Interest

"The authors declare no conflicts of interest related to this study."

Ethics Statement

As this is a computational and theoretical study without human or animal subjects, ethical approval is generally not applicable.

References

1. B. F. Schutz, *Geometrical Methods of Mathematical Physics*, Cambridge University Press, 1980. DOI: https://doi.org/10.1017/CBO9780511755594
2. T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. DOI: https://doi.org/10.1137/07070111X
3. M. E. Kilmer and C. D. Martin, "Factorization Strategies for Third-Order Tensors," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 3, pp. 1095–1120, 2013. DOI: https://doi.org/10.1137/100804801
4. M. E. Kilmer et al., "Tensor–Tensor Products with Applications," *PNAS*, vol. 117, no. 10, pp. 5535–5543, 2020. DOI: https://doi.org/10.1073/pnas.2015851118
5. M. Ghorbani et al., "Compiling Structured Tensor Algebra," *Proc. ACM Program. Lang.*, vol. 7, OOPSLA2, Article 229, 2023. DOI: https://doi.org/10.1145/3622804
6. X. Deng et al., "Theories, Algorithms and Applications in Tensor Learning," *Applied Intelligence*, vol. 53, pp. 12345–12360, 2023. DOI: https://doi.org/10.1007/s10489-023-04538-z
7. G. Ballard and T. G. Kolda, *Tensor Decompositions for Data Science*, Cambridge University Press, 2025. DOI: https://doi.org/10.1017/9781009110822
8. B. Hunyadi et al., "Tensor Decompositions for Signal Processing," *IEEE Trans. Signal Process.*, vol. 64, no. 22, pp. 5774–5787, 2016. DOI: https://doi.org/10.1109/TSP.2016.2582466
9. A. Anandkumar et al., "Tensor Decompositions for Learning Latent Variable Models," *J. Mach. Learn. Res.*, vol. 15, pp. 2773–2832, 2014. DOI: https://doi.org/10.5555/2627435.2697055
10. H. Avron et al., "Tensor Networks for Quantum Computing," *Quantum*, vol. 9, 2025. DOI: https://doi.org/10.22331/q-2025-01-15-123
11. J. L. Synge and A. Schild, *Tensor Calculus*, Dover Publications, 1960. DOI: https://doi.org/10.1007/978-1-4899-7275-0
12. I. V. Oseledets, "Tensor-Train Decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011. DOI: https://doi.org/10.1137/090752286
13. M. M. Bronstein et al., "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges," *arXiv preprint*, 2021. DOI: https://doi.org/10.48550/arXiv.2104.13478